AMENDMENTS TO THE SPECIFICATION

Please amend page 3, lines 12-16 as follows:

Various paradigms exist as a result of the need to standardize the methods by which objects can be distributed and accessed over a network. These are Microsoft Corporation's Distributed Component Object Model (DCOM), JavaSoft's ~~Java~~JAVA™ (hereafter, JAVA)/Remote Method Invocation (~~Java~~JAVA/RMI), and Object Management Group's Common Object Request Broker Architecture (~~CORBA~~CORBA™, hereafter CORBA).

Please amend page 3, line 18 – page 4, line 10 as follows:

Though some differences are present among these models, they principally work in the following way. Objects that provide services are typically located on servers. These objects are queried by applications running on clients using a specified data communication transport layer protocol--the Object Remote Procedure Call (ORPC) for DOOM; the ~~Java~~ JAVA Remote Method Protocol (JRMP) for ~~Java~~JAVA/RMI; and the Internet Inter-ORB Protocol (IIOP) for CORBA. A client suitably formats a query message in the appropriate protocol language and transmits the query message, which is routed to the appropriate server, whereupon it is executed, and a response message is formatted and routed back to the client. As referred to herein, the term "object" may mean the object definition, associated operations, attributes, etc., and implementation for that object. As will be appreciated by those of skill in the art, at times the term "object type" is used to refer to the definition of the operations and attributes that software external to the object may use to examine and operate upon the object. The "object type" is also known as the "interface." Also, the term "object," may be used to refer to an actual run-time instance of an object and will be made clear by the context.

Please amend page 4, lines 12-20 as follows:

A server configured to be a ~~Java~~JAVA/RMI server comprises objects that have predefined interfaces, which can be used to access the server objects remotely from another machine's ~~Java~~ JAVA Virtual Machine (JVM). A ~~Java~~JAVA/RMI server object interfaces declare a set of methods that indicate the services offered by that server object. A program resident on the server called an RMI Registry stores and makes available to clients information about server objects. Typically, a client object obtains information regarding the methods and other properties of a server object by performing an operation such as "lookup" for a server object reference. This lookup typically works by the client object specifying an address in the form of a Universal Resource Locator (URL) and transmitting the address to the server's RMI Registry.

Please amend page 5, lines 1-4 as follows:

The clients and servers also include interceptors. The interceptors provide hooks to programmers to execute their piece of code at certain points during ~~ORB~~ORB™ (hereafter, ORB). Typical uses of the interceptors include: transaction service integration, security message compression and encryption, fault tolerance and other operations such as tracing, profiling, debugging, logging.

Please amend page 6, lines 10-19 as follows:

Object adapters such as the CORBA-standard Portable Object Adapter (POA) mediate between an ORB and a set of programming language servants. In general, though there could be many instances of POAs to support CORBA objects of different ~~2styles~~styles, and all server applications have at least one POA called the *Root* POA. Each POA instance represents a grouping of objects that have similar characteristics. These characteristics are controlled via POA policies that are specified when a POA is created. The Root POA, which is present in all server applications, has a standard set of policies. POA policies are a set of objects that are used to define the characteristics of a POA and the objects created within it. The CORBA standard specifies that interfaces for POA, POA manager (which is a class to manage multiple POAs) and the POA policies are defined in a standard module.

Please amend page 8, lines 6-10 as follows:

Composite applications combine new business logic with existing logic, processes, and data to meet business needs. A new system is required to provide all of the components of an enterprise portal infrastructure. This new set of server-based products ~~2marshals~~ marshals the company's expertise in design and support of enterprise systems from diverse applications using object and component technologies.

Please amend page, 14, line 16 – page 15, line 3 as follows:

The verb *bind* means "to tie". As discussed above this section is directed to describing tying together the applications that make up a distributed system. In particular, the verb *binding* can be defined as the process of establishing a channel over which communications can take place, and the noun *binding* as the realization of such a ~~chanel~~channel. In CORBA, the binding process is initiated by a client application attempting to make an invocation using an object reference for which no appropriate binding already exists. The target object reference identifies the server application with which a binding will be established. The binding that is established represents the negotiated protocol and services that allow the client to make invocations on the target object, and any shared state required by the protocol and services at the client and server.

Please amend page 15, lines 5-9 as follows:

The ART binding process can be modeled as a black box with inputs and outputs. The inputs are the target object reference and the client-side policies that constrain the binding process. The outputs are the selection of client-side and server-side ORB ~~2services~~services, including the transport protocol, that make up the binding, plus any binding-specific state that they maintain. This section describes the following aspects of this process and its results:

Please amend page 47, lines 7-8 as follows:

The read-only ~~orb~~ ORB attribute provides access to the **CORBA: : ORB** instance that received the request. Its accessor can be called at any time, and the result is not released.

Please amend page 55, lines 7-9 as follows:

The ~~orb~~ ORB and **object_key** attributes behave identically to these attributes of the **ServerRequestData** interface. The **thread_context** attribute is also substantially similar, except that there are no **ServiceContextLists** available to propagate service-specific state.

Please amend page 111, lines 10-11 as follows:

Figure 7 illustrates the use of flyweight patterns **706** for intrinsically chained interceptors **702** (in conjunction with state information ~~708~~704) as will be discussed in greater detail herein.